Intelligent Motion Detection and Compressed Sensing

New Camera Technology with New Challenges

Our project investigates intelligent motion detection using compressed sensing (CS), an emerging data acquisition technology with promising applications for still and video cameras. CS incorporates image compression into the initial data collection on an image rather than generating a compressed file after initially collecting a larger amount of data. By taking only as many data points as will be stored or transmitted, compressed sensing seeks to eliminate the waste from collecting many, many pixel-intensity values on an image and then using compression algorithms (such as JPEG or GIF) to encode a much smaller number of data points to closely approximate the information in the original image. [1]

Lower resource usage makes compressed sensing cameras attractive choices for low-power applications including security cameras. Ilan Goodman, Ph.D candidate at Rice University, has demonstrated that motion detection using a simulated CS camera is possible by computing entropy changes between successive CS measurements [2]. Starting from his work, we explore what can be determined about the motion of an object using compressed sensing.

[1] "Compressed Sensing Resources." Digital Signal Processing Group, Department of Electrical and Computer Engineering, Rice University. 2005. http://www.dsp.ece.rice.edu/cs.

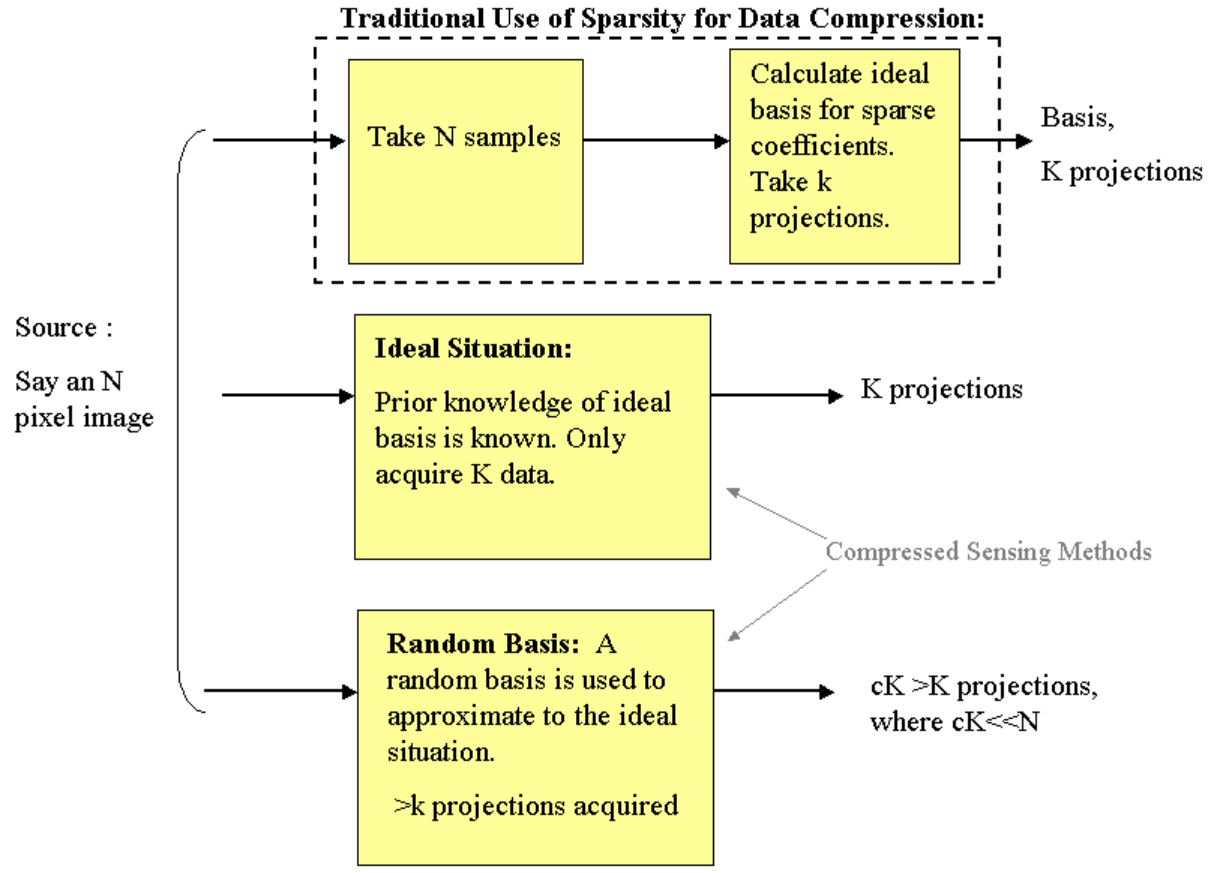
[2] I.N. Goodman & D.H. Johnson. "Look at This: Goal-Directed Imaging with Selective Attention." (poster) 2005 Rice Affiliates Day, Rice University, 2005.

Compressed Sensing

Compressed sensing is based on exploiting sparsity. Sparse signals are those that can be represented as a combination of a small number of projections on a particular basis. (This new basis must be incoherent with the original basis.) Because of sparsity, the same signal can be represented with a smaller amount of data while still allowing for accurate reconstruction.

In non-compressed sensing methods, one would first aquire a large amount of data, compute an appropriate basis and projections on it, and then trasmit these projections and the basis used. This is wasteful of resources since many more data points are initially collected than are transmitted. In compressed sensing, a basis is chosen that will approximately represent any input sparse signal, as long as there is some allowable margin of error for reconstruction.

Comparison of Data Aquisition Algorithms that Use Sparsity



Comparison of different algorithms. Our project focuses on the third algorithm using random basis projections.

The pre-defined basis for the optimal case (as represented in the block diagram) can only be determined with prior knowledge of the signal to be aquired [1]. However, in practical applications such information is not usually known. To generalize to a basis that gives sparse projections for all images, a random basis can be used. A matrix of basis elements is generated from random numbers such that the basis elements are normal and orthogonal on average. Since using projections on a random basis is not the optimally sparse case, a larger number of projections must be taken to allow for reconstruction [2],[3]. However, this number is still far fewer than the number of datapoints taken using the traditional approach which exploits sparsity after data acquisition.

One application of compressed sensing is an N-pixel camera being designed by Takhar et al. which acquires much fewer than N data points to record an image [4].

For a more detailed explanation of compressed sensing, please refer to the literature on http://www.dsp.ece.rice.edu/cs

[1] D. Baron, M. B. Wakin, S. Sarvotham, M.F. Duarte and R. G. Baraniuk, “Distributed Compressed Sensing,” 2005, Preprint.

[2] E. Candes, J. Romberg, and T. Tao, “Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information,” IEEE Trans. Inform. Theory, 2004, Submitted.

[3] D. Donoho, “Compressed sensing,” 2004, Preprint.

[4] D. Takhar, V. Bansal, M. Wakin, M. Duarte, D. Baron, K. F. Kelly, and R. G. Baraniuk, “A compressed sensing camera: New theory and an implementation using digital micromirrors,” in Proc. Computational Imaging IV at SPIE Electronic Imaging, San Jose, January 2006, SPIE, To appear.

Can Random Noise Yield Specific Information?

The novel challenge with using random basis projections for intelligent motion detection is that there is no spatial information about the image or movie in the compressed sensing data. Traditional pixel-based cameras provide a graph of light intensity over position and, logically, most pixel-based detection approaches use the information about where the motion occurs to help classify it. [1] The CS data provides us not with intensity at a point, but with the similarity (inner product) between the original pixel image and a selection of basis elements composed of random noise spread throughout the image plane. Intelligent detection for CS, therefore, must use approaches radically different from detection used on conventional video.

Simplicity for Low Power

A key feature of CS systems is the potential for extremely low power consumption. To keep the overall power of the system low, any computations we perform must be low power as well. For practical application, the algorithms chosen must be simple to compute.

Investigation Goals

Working in a very new field, we began our project with the open-ended goal of researching what types of motion can be detected using compressed sensing and of implementing at least one such measurement. After much deliberation and trial, we chose to implement speed detection, specifically calculating the speed of a known object along the direction of motion. (Different shapes complicate the problem, as we will discuss.) Looking ahead to more sophisticated detection, we also wanted an extensible system to use the results of several relevant calculations to automatically characterize motion.

Though beyond the scope of our investigation, our motivation is a camera running our intelligent detection system capable of determining if a desired

type of motion is observed. We pursue algorithms which could be implemented in real time at the data collection point.

[1] For some interesting work in intelligent detection with traditional cameras, see the work of James W. Davis and Aaron F. Bobick out of the MIT Media Laboratory in the late 1990s.

Methodology for Extracting Information from "Random" Measurements

Simulating Compressed Sensing

Because compressed sensing cameras are not yet availiable, we used a Matlab routine written by Ilan Goodman to simulate CS measurements from a standard pixel image file [1]. Only the compressed sensing measurements are passed into our suite of calculation programs which run exactly as if the CS measurements came from a hardware-implemented CS camera.

To implement compressed sensing on an image (matrix) according to the definition, a random matrix the same size of the image is generated. The projection (inner product) of the image onto the random basis matrix gives a single compressed sensing measurement. This is repeated with different (fixed) random matricies until the desired compressed sensing resolution is achieved. This is computationally intensive and so a different approach is used in practice to simulate our data: first, every pixel of the image is randomly mapped to a different location to randomize the image. Next, the DCT (discrete cosine transform) is taken on the randomized image. This process of randomization and projection is equivalent to projection onto a random basis [1].

Random, On Average

We exploited two key facts about compressed sensing on a random basis to calculate speed:

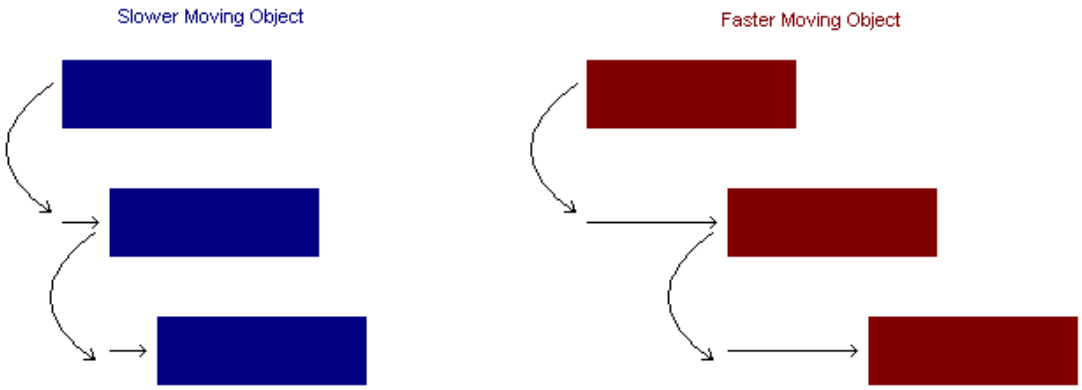
1) The average value of the elements of the random basis used is 1.

2) On a given image, a fixed random basis yields the same projections every time it is used.

While seemingly trivial, this basic data allows us to determine velocity quite accurately based on a few observations in the pixel domain.

Consider the following moving rectangles:

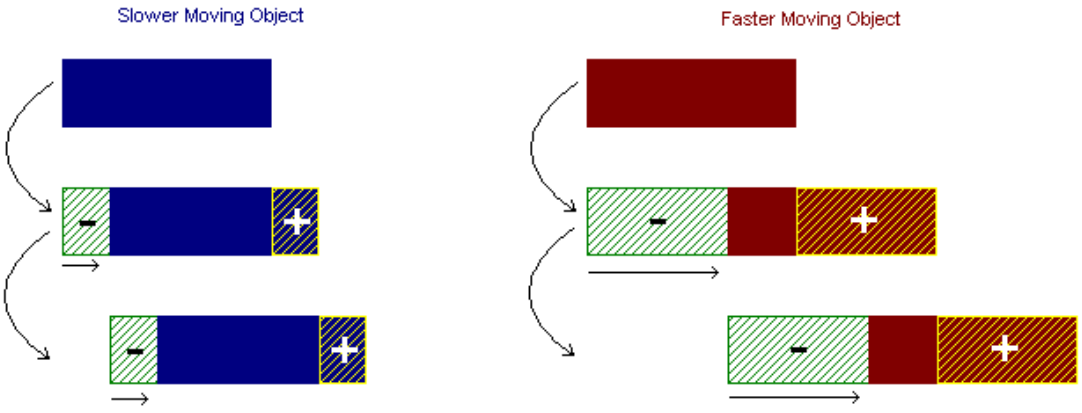
Rectangles Moving Horizontally at Different Speeds



Two rectangles moving to the right with constant speeds

Now consider the difference between subsequent frames showing the motion of the rectangles:

Moving Rectangles: the Difference Between Subsequent Frames



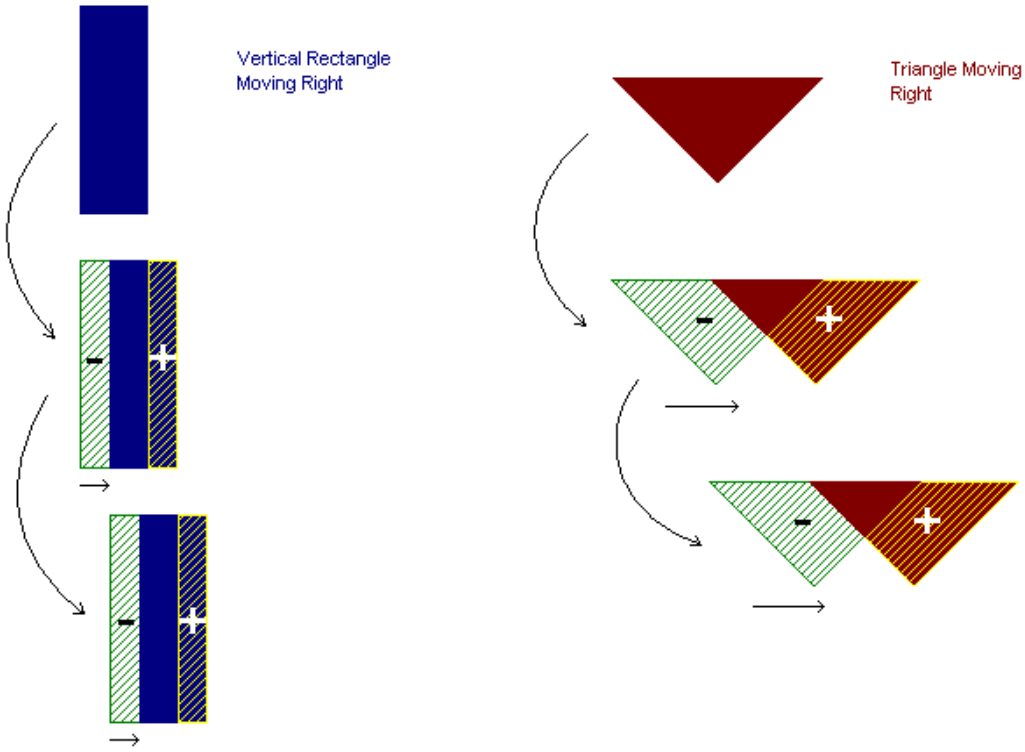
In each subsequent frame, areas where the current and previous rectangles overlap remain the same while new area is added in the direction of motion and old area is lost opposite the direction of motion.

Since the red rectangle is moving faster, there is less overlap between subsequent frames and more area is both added to and subtracted from the image area. Therefore, we would expect that since the difference between subsequent images is greater for the red rectangle that the difference between consecutive compressed sensing projections along the same basis element is also greater. Taking a simple difference between consecutive CS measurements should yield a measure of the change between frames.

This basic intuition can also be supported rigorously. The difference between frames can be thought of as an image itself with a positive region on the leading edge of motion and a negative region at the trailing edge. Since the CS measurement process is linear time invariant assuming a fixed basis, the difference between projections in subsequent frames is the same as the projection of the difference image [2]. If the background behind the moving objects has zero value, then the CS projection values of the difference image is based solely on the difference between frames of the original images. The larger the non-zero area of the difference image, the larger the inner product with the CS basis elements are expected to be and a positive relationship exists between speed and frame difference measured from the compressed sensing data.

These calculations yeild a ratio between the change between subsequent frames along the direction of motion with respect to the total intensity in the frame. The same shape either moving in a different direction or with different orientation will produce different results. For more complicated shapes, the amount of change is not linear with speed and we expect the measurement of change will be more complicated, but still deterministic.

Difference Images for Other Objects



Different objects, or objects shaped differently with respect to the direction of motion, produce differing overlap areas between subsequent frames.

Resolution Limit

Calculating velocity in this way is limited to sampling rates that show overlap between consecutive frames. If the object is moving so quickly that there is no overlap, it is unclear how far it has moved: a speed where the frames are just slightly discontinuous will give similar results to a much faster speed.

[1] I.N. Goodman & D.H. Johnson. Look at This: Goal-Directed Imaging with Selective Attention. (poster) 2005 Rice Affiliates Day, Rice University,

2005.

[2] Goodman, I.N. Personal conversation. 9 December 2005.

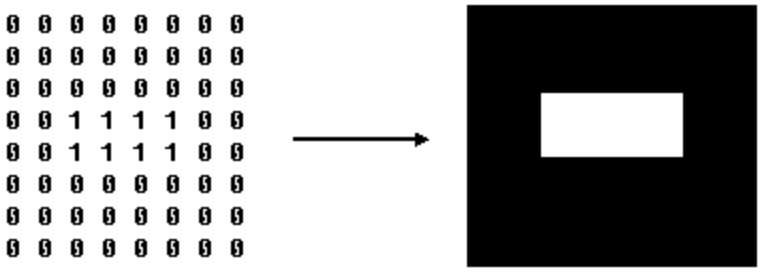
Idealized Data for Motion Detection

The main purpose of our device is to detect velocity of motions in a movie file, and it is helpful to understand the mechanism in idealized cases. It is too complicated to come up with appropriate algorithms for all possible input data at once, and we want to look at the simpliest cases first. Our choice of data is a simulated movie. Reducing to bare bones, a blank, constant background(noise-free) with a single object moving across the screen is a good start.

Making Frames

Manual coding of movies was done in MATLAB, using matrices of zeros and ones. zeros represents background, and ones represent an object. For simplicity, movies are in grayscale (black for zeros and white for ones).

Example:
Matrix To Frame



Command 'imagesc(matrix)' on MATLAB will give us a simple picture

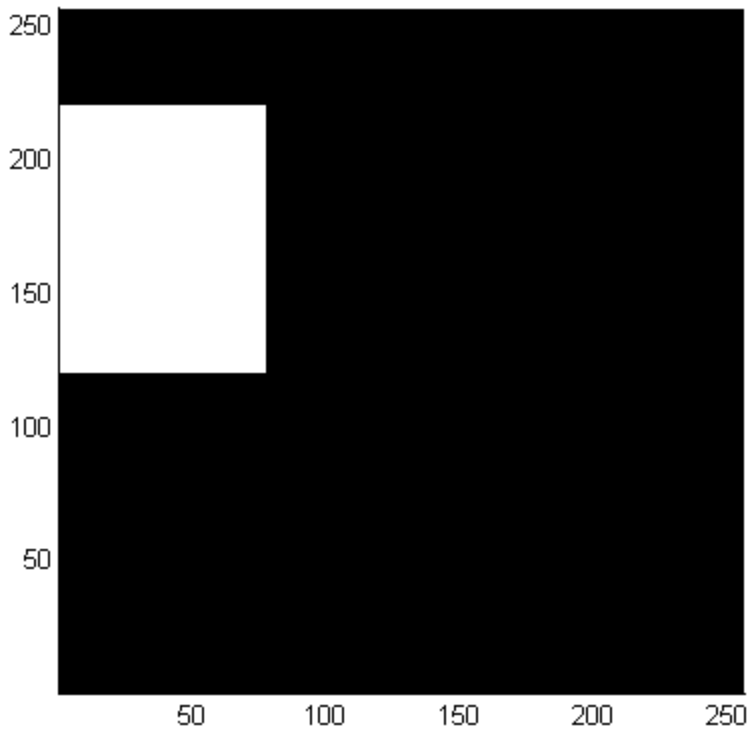
Looking at above matrix, different shapes can easily be made by arranging the ones in different layouts. We have created square, vertical rectangle, horizontal rectangle, triangle, and circle shapes. Once a desired image with an object is created, the 'getframe' command in MATLAB converts images to frames.

Making Movies

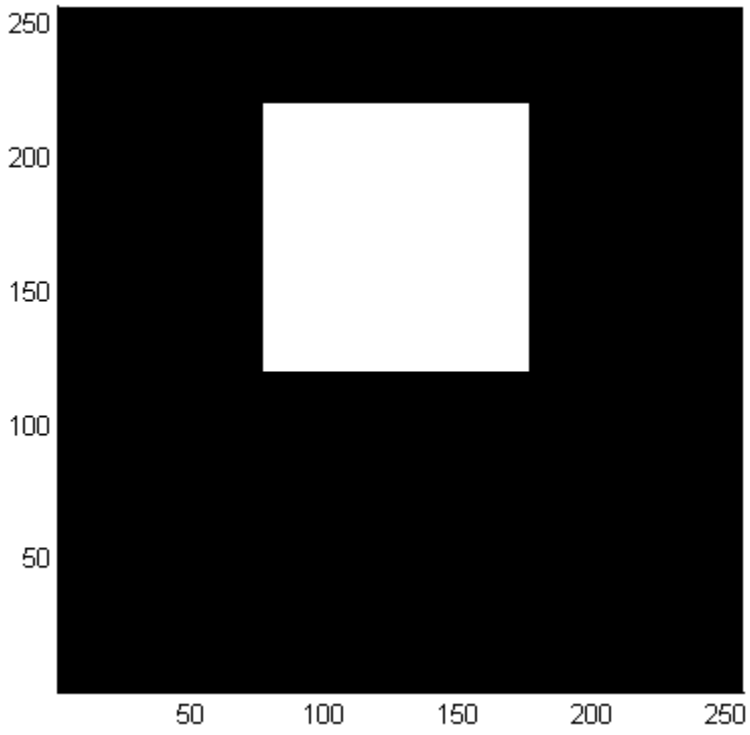
To make a movie of a "moving" object (represented as a group of ones), our program can simply create multiple frames with the group of ones in different locations. One can move most any object in any direction in a reasonable frame size. However, creating a movie with non-axial motion can be very tedious in MATLAB, so we chose motions along the horizontal and vertical axis.

Note: For our purpose, we set the screen size to be 256x256. Each pixel corresponds to one entry in the matrix.

Example:
Moving Square



Square moving into the screen



Square in the screen

Different velocities can be achieved by varying number of pixels moved over per frame. In other words, within the same number of frames, a movie with a faster object will have its object moved over further across the screen than one with a slower object.

Example:
Moving Triangle With Different Speeds

frame 2

frame 3

Triangle moving at 10 pixels per frame

frame 2

frame 3

Triangle moving at 50 pixels per frame

Speed Calculation: the Details

Extensibility

To make our speed algorithm as extensible to as many different situations as possible, we normalize the results with respect to the overall intensity in the image. This gives a measure of change whose magnitude is scaled by the size of the image. The overall intensity is calculated as where Cn,t is the compressed sensing measurement along basis element n and time t. The CS resolution, or number of data points taken per frame, is N.
Equation:

—

Average Absolute Change to Measure Speed

As described earlier, increased velocity yields greater change between subsequent measurements in time of a particular projection. Our calculation to measure velocity is:
Equation:

— —

Since we are interested only in the amount of change that occurs, the absolute value is taken. Because the projections themselves are random, any one measurement is not a good indication of the change and it is only by averaging a number of calculations that a meaningful value can be computed.

Average Squared Change to Measure Speed

As another way to measure the change between frames, the squared difference between frames is taken. This measure was suggested by Ilan Goodman to explore the Parseval equivalencies often observed between domains.

Equation:

Since this metric computes the square of the two norm of , it is linearly proportional to the intensity changes of the difference image in the pixel domain.

Ability to Detect Speed: Results

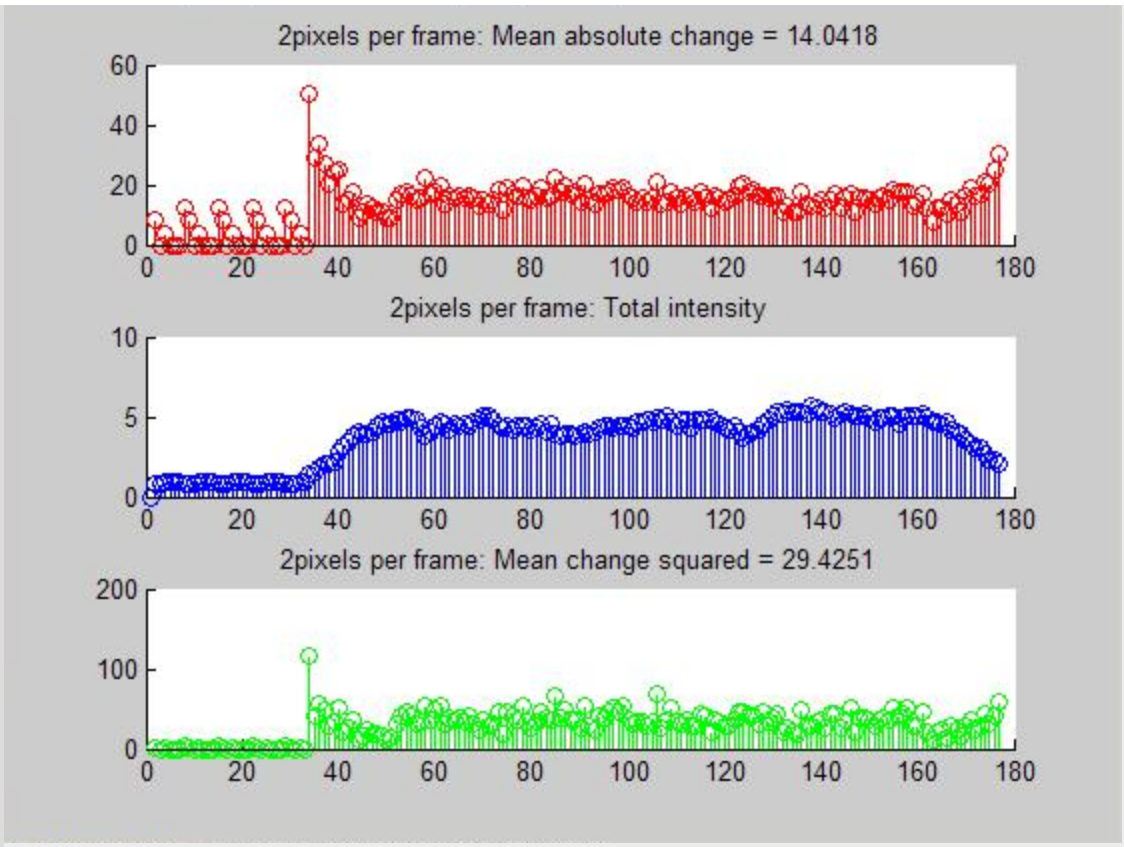
The results of running our intensity and average change calculations on CS data on objects moving at different speeds shows that a fully deterministic relationship exists between our calculations and the speed of the object along the direction of motion.

Recall our three calculations: Total intensity is a measurement of non-zero area in a single frame. Mean absolute change is the absolute value of the average change between basis projections in subsequent frames. Mean change squared is the mean of the squared difference between basis projections in subsequent frames.

Calculations Performed on Each Movie Clip

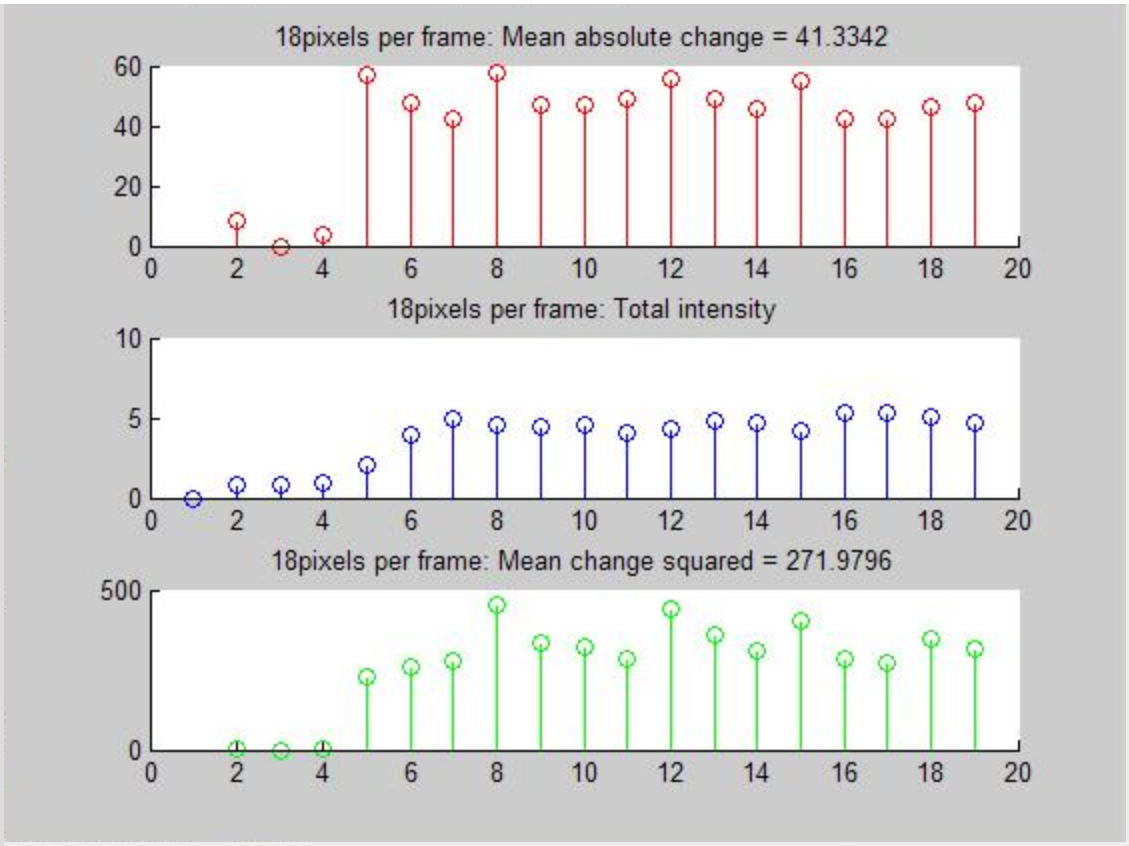
The three calculations were computed for every frame of a movie clip. Sample results of the metrics over time are shown below.

Example:
Mean Magnitude of Coefficient Differences (2pixels/frame)



These metrics were extracted from a movie of a standing rectangle moving uniformly across the screen at 2 pixels per frame

Mean Magnitude of Coefficient Differences (18pixels/frame)



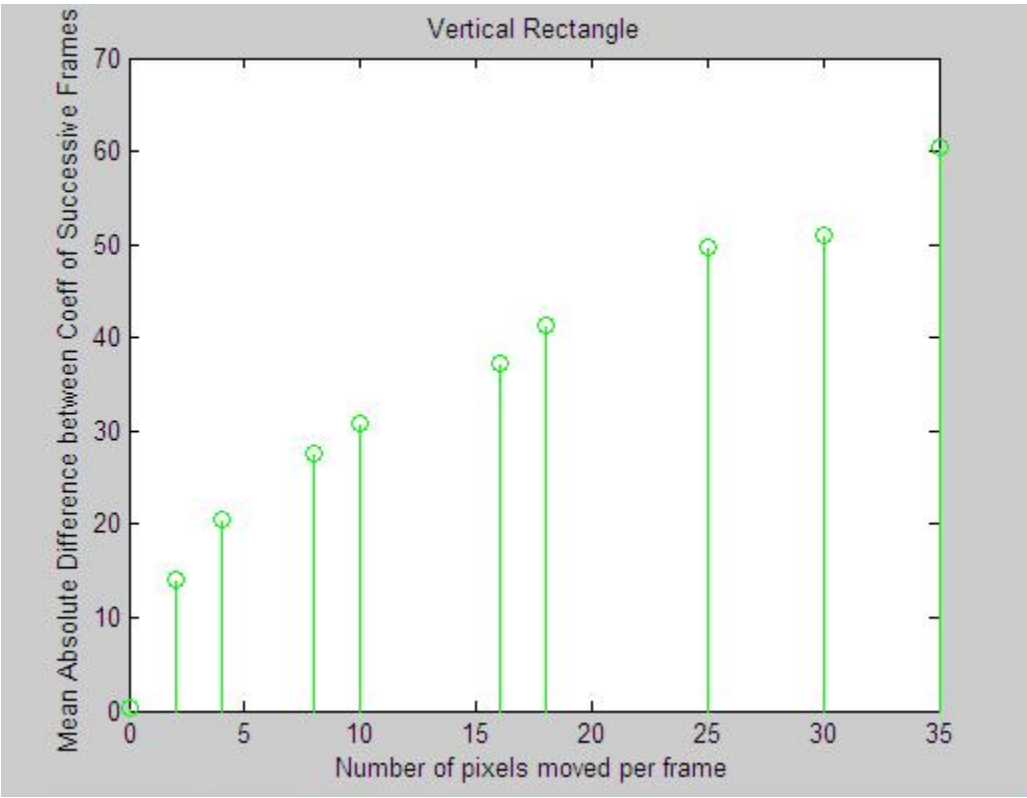
These metrics were extracted from a movie of a rectangle moving uniformly across the screen at 18 pixels per frame. The horizontal axis in each graph is time.

Since the video clips analyzed contained an object moving at constant velocity, it is expected that the average absolute and squared changes would be constant across a frame. The noise in the data comes from the random nature of the basis used.

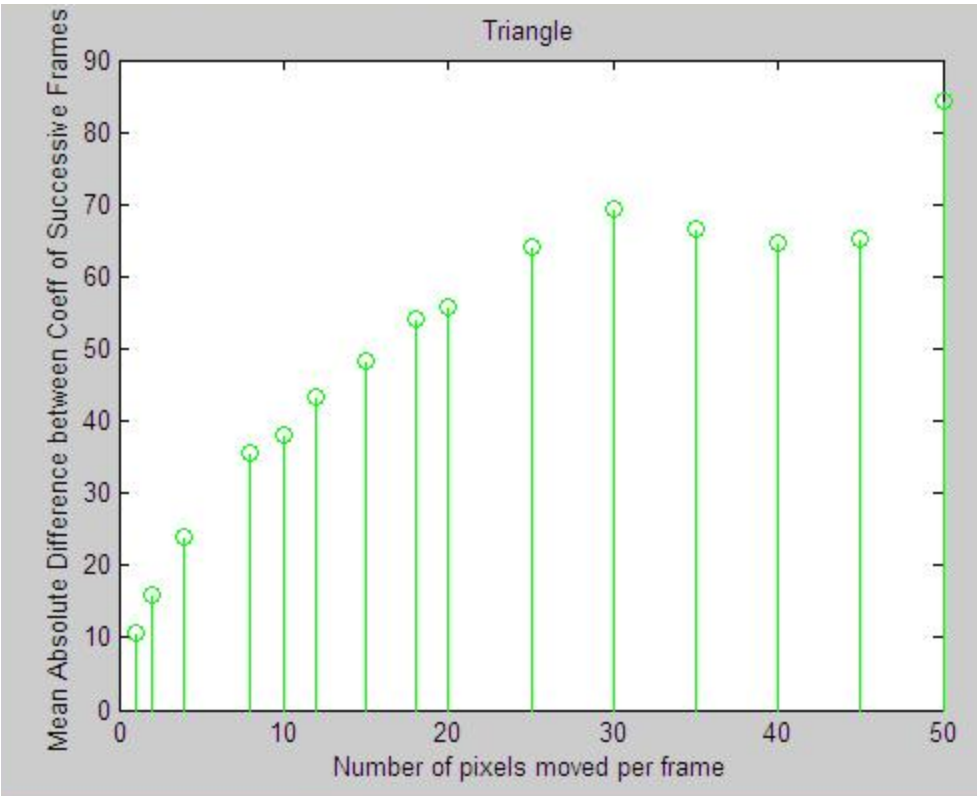
Velocity Trends

To plot average mean and absolute change with respect to velocity, the average value of each calculation is taken from the above data. The results are graphed below.

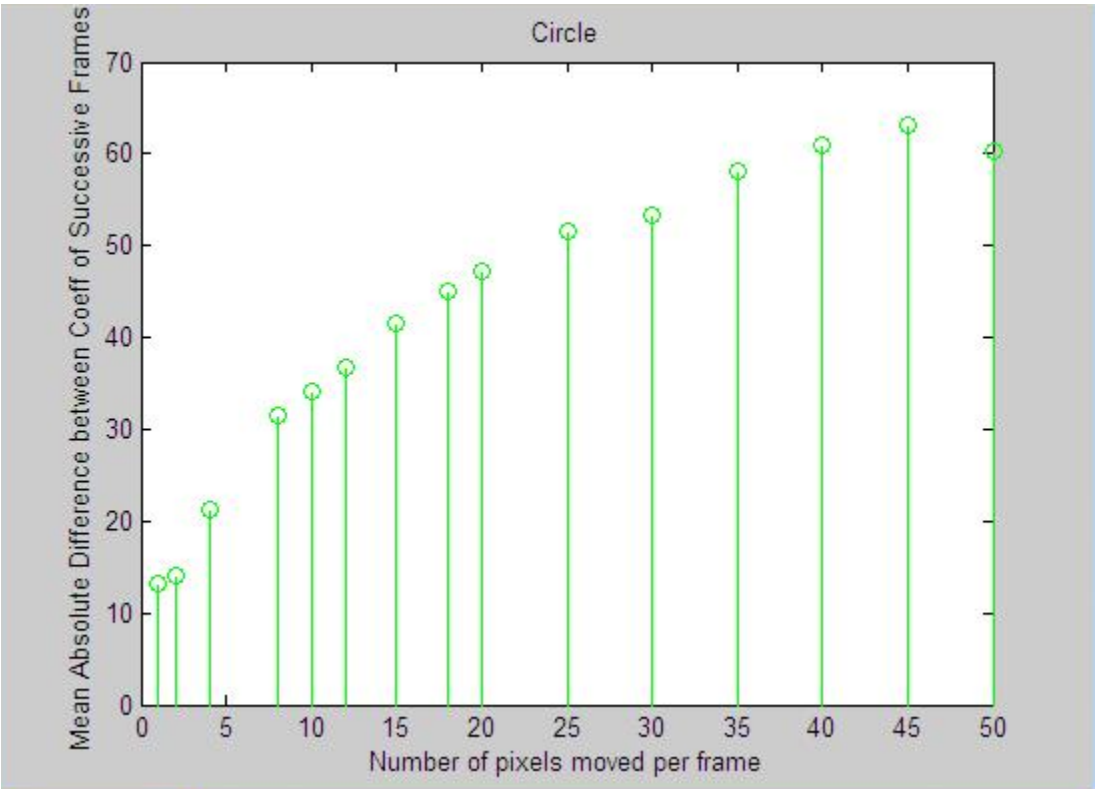
The first three graphs show variations in absolute mean differences with respect to velocity for different shapes. Each shape has a unique curve because of the shape of image overlap for different speeds. As future work, these minor differences could allow us to distinguish shapes from such data. The trends observed here are not linear, but a fit curve can easily be generated from a few data points and used to classify new data. Since the calculation of this feature is very simple, it could be implemented in low power applications.



Absolute Average Change (AAC) vs. pixels/frame



Absolute Average Change (AAC) vs. pixels/frame

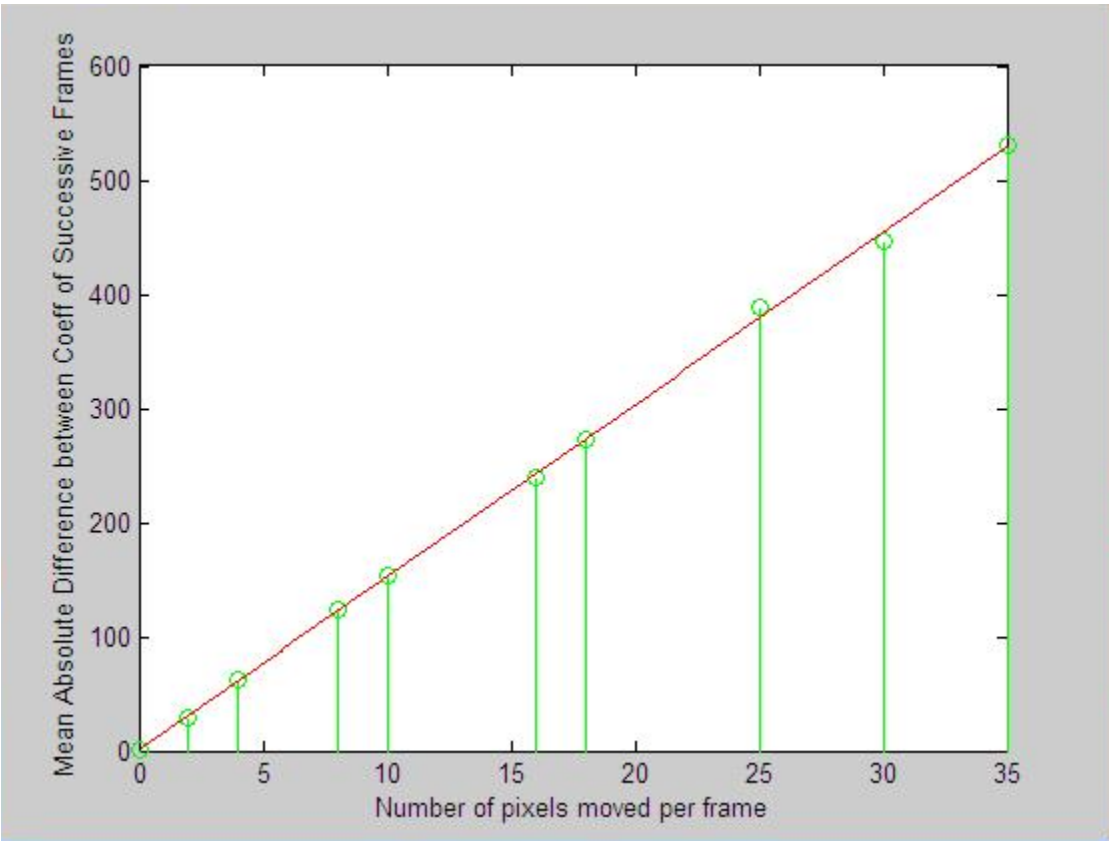


Absolute Average Change (AAC) vs. pixels/frame

Linear Relationship

With help from Ilan Goodman, we saw that Parseval's theorem dictates that the two-norm of the change in area is linearly proportional to the two-norm of the change in the compressed sensing coefficients. Since the change in area is linear in velocity for rectangles, this predicts that the average squared change will be linear for a rectangle. This relationship is supported by our data.

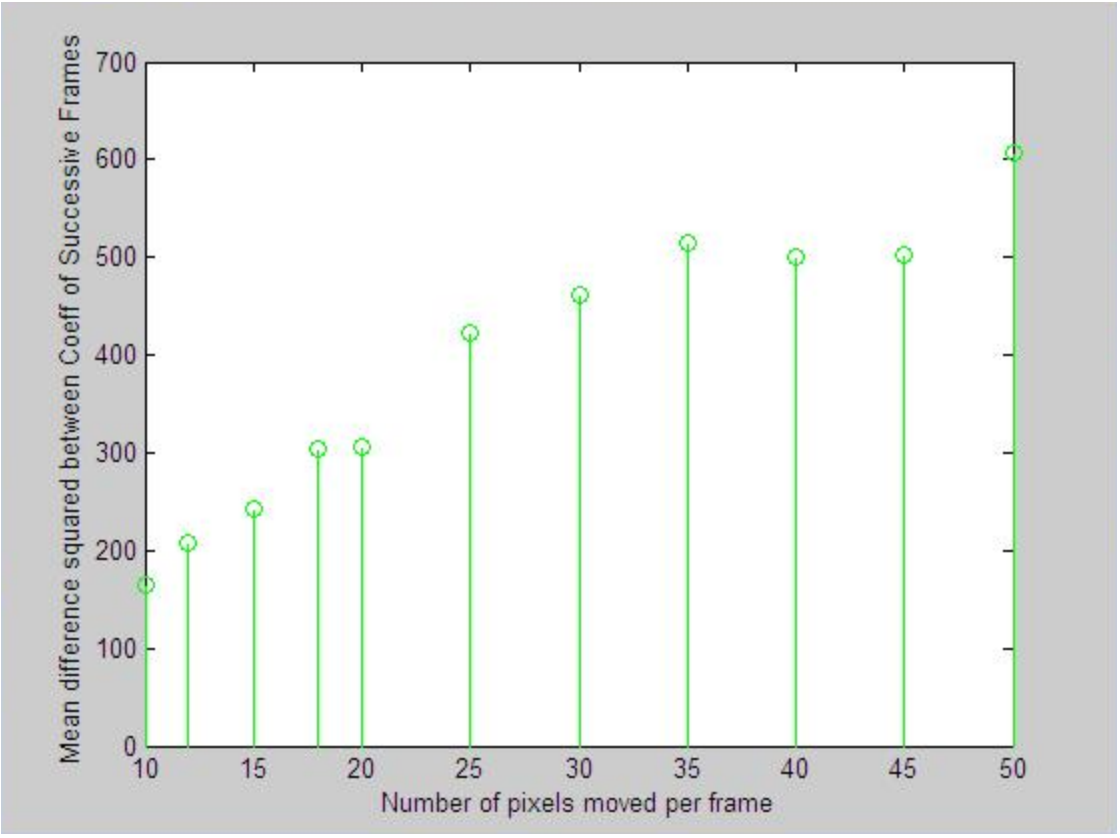
Mean Change Square for Vertical Rectangle



As predicted by Parseval's, we observed a linear relationship in the Average Squared Change (ASC) plot

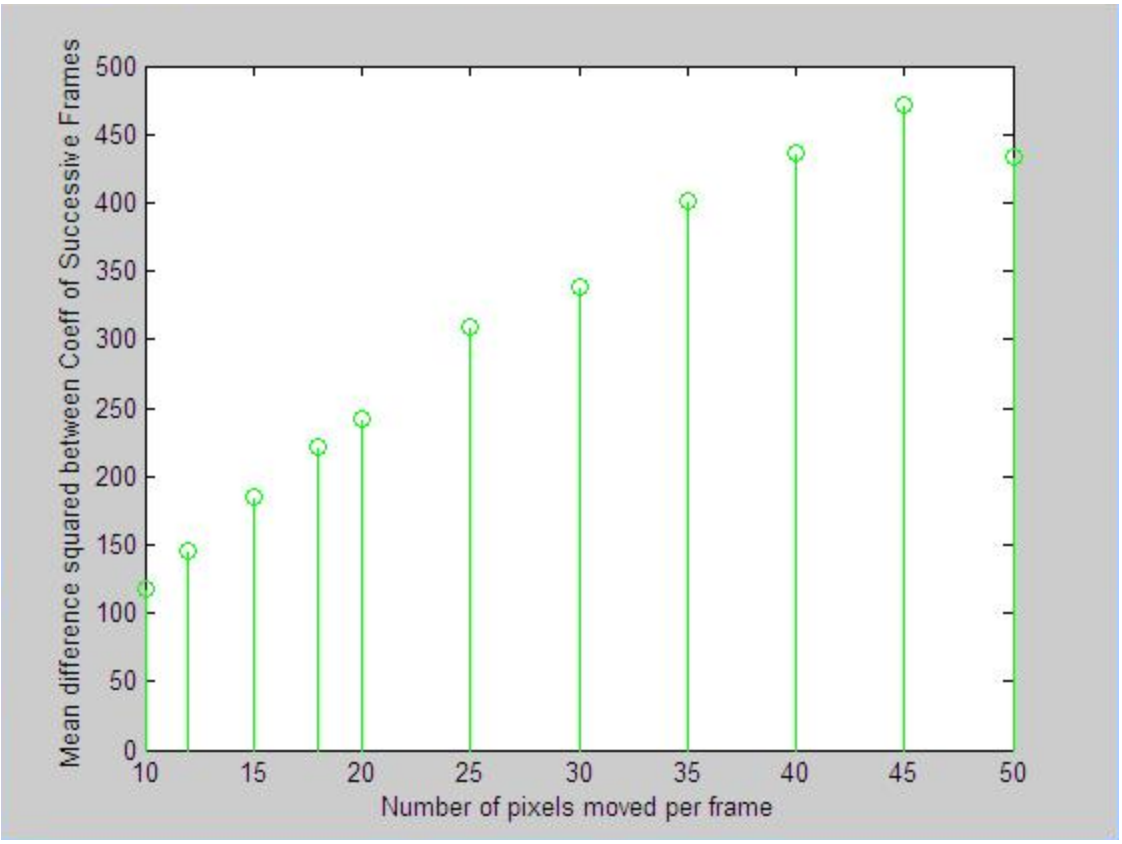
The linear relationship does not hold for other shapes because the overlap between frames is not linear in velocity.

Mean Change Squared for Triangle



For a triangle, we see a non-linear dependency on ASC curve.

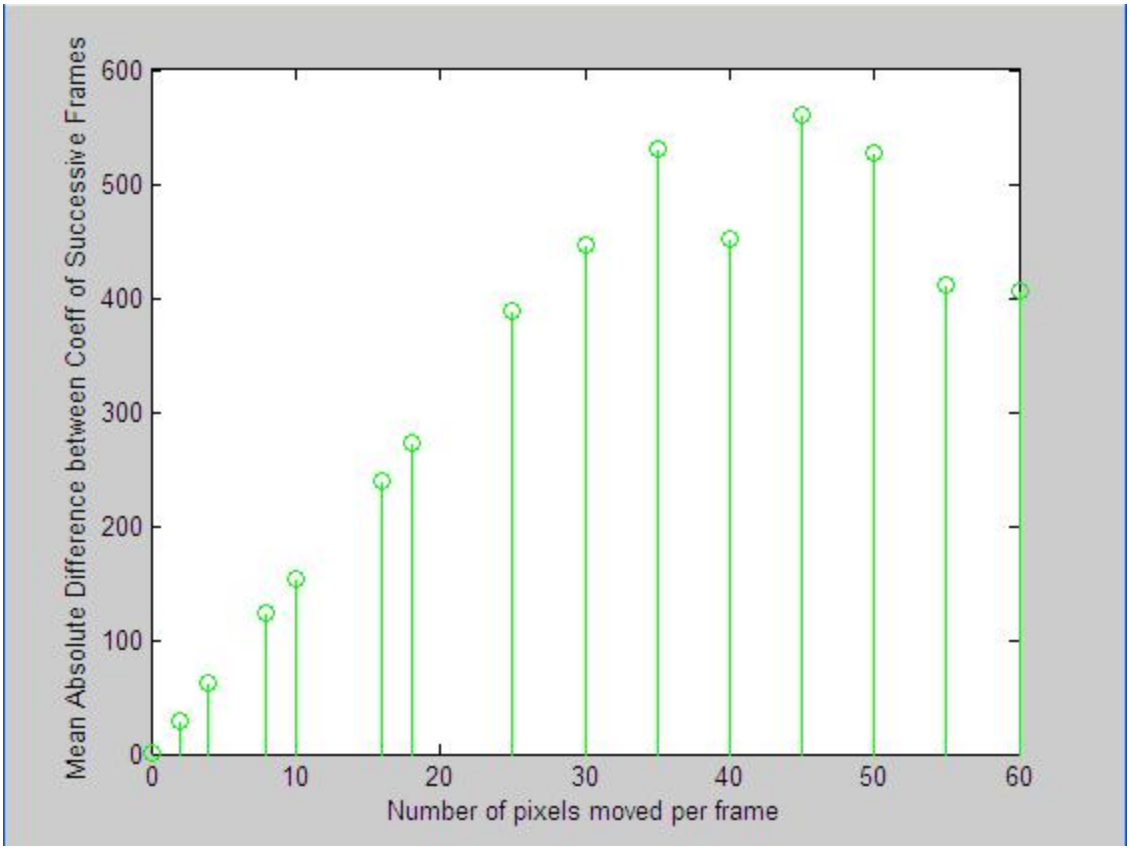
Mean Change Squared for Circle



A circle does not show a linear dependency on ASC curve either.

The velocity of a known object can be determined using a fit of this data for any shape object. However, there exists an upper bound to the velocities that we can detect. Consider an object of size x pixels along the direction of motion. Once the velocity exceeds x pixels per frame, the amount of overlap has already gone to zero and the exact velocity cannot be determined. This can be described as analogous to aliasing.

Velocity Detection Upper Limit



Above the velocity resolution limit of 35 pixels/frame, the velocity cannot be determined. The data for high velocities contains fewer datapoints and has a greater deviation.

Concluding Remarks for CS Motion Detection

The most important result that we have obtained is that the speed of a known moving object in the direction of motion can be extracted from the compressed sensing data using a very small number of coefficients, of a size less than 1% of a corresponding pixel image. Despite taking only fifty or a few hundred measurements from the set of random basis coefficients, the speed information is still present and can be extracted.

However, there is a resolution limit on the speeds that can be detected for the moving objects. As discussed in the analysis, when there are no overlapping pixels in successive frames for the same data, there is no way to distinguish between speeds above this threshold.

We use Parsevals Theorem to explain the linear relationship of the average squared change and velocity for a moving rectangle.

Future Work in CS Motion Detection

There are many future directions that we can foresee for intelligent motion analysis on compressed sensing data due to the minimal computation and data storage required.

One major point of future study is the feature extraction algorithms for compressed sensing. Our analysis looked at only a very small part of the many features that we think can be gleaned from the compressed sensing measurements. For example we can look at the acceleration, which could be trivial using the difference in the velocity feature. Our calculation of total intensity can also be fully exploited. Similarly, we believe it is possible to develop algorithms for shape recognition.

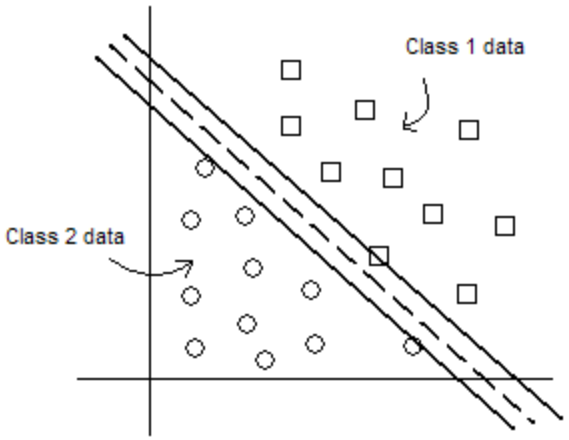
When the number of feature calculations availiable increases, a Support Vector Machine (SVM) can be used to analyze the data. The SVM is a prediction algorithm which after being trained with sufficient data can classify new data on the basis of the training. This is very useful when the data is multi-dimensional.

Support Vector Machines

A Support Vector Machine (SVM) is a decision-based prediction algorithm which can classify data into several groups. It is based on the concept of decision planes where the training data is mapped to a higher dimensional space and separated by a plane defining the two or more classes of data [1].

A simple example is seen in Figure 1. Squares are data of class one while circles are data of class two. The SVM sets up the decision plane (in this case a simple line) and separates the two classes.

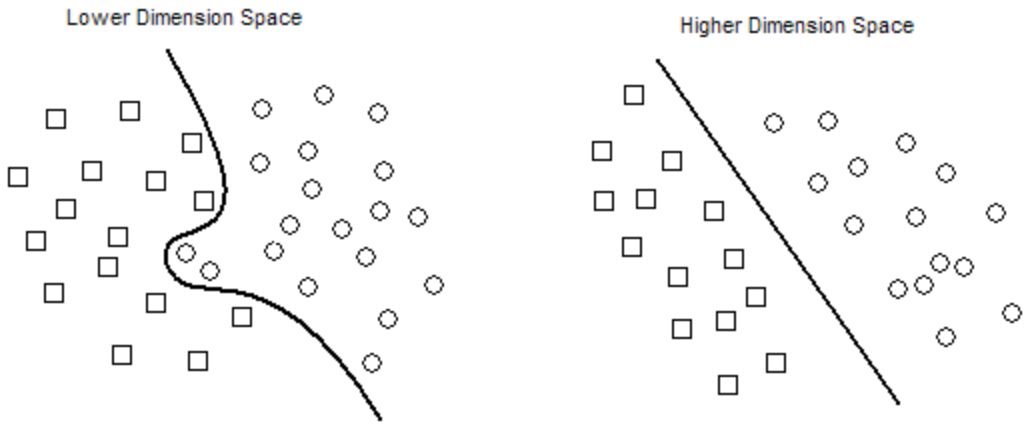
2-D Example



A simple 2-D example for the decision algorithm in an SVM

However, often the data is not distinguishable in two dimensions in which case it is mapped to higher dimensions and the same process is done. An example is shown in Figure 2.

Mapping In Higher Dimensional Space



When there is no solution in a lower dimension it can be mapped to a higher dimension and a decision plane is easier to construct

Support Vector Machine models can be classified into four major groups.

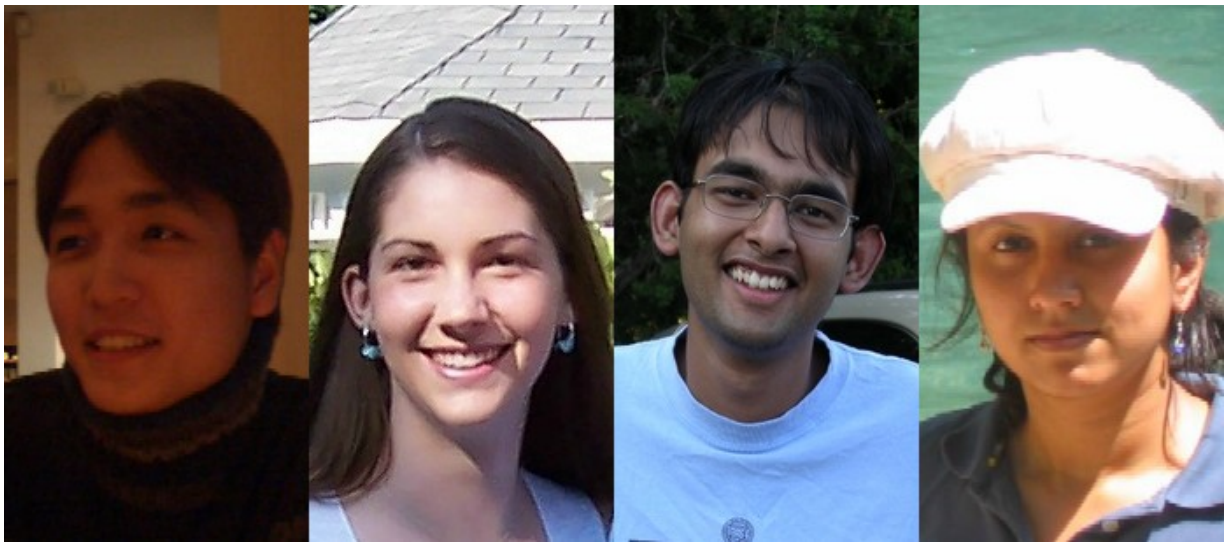
1. C-SVM classification
2. nu-SVM classification
3. epsilon-SVM regression
4. nu-SVM regression

The first two are classification algorithms which minimize different error functions while the second two perform similar algorithms by regression [1].

[1] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

The Team and Acknowledgements

The Team



Grant Lee, Heather Johnston, Siddharth Gupta, Veena Padmanabhan

Grant Lee, Heather Johnston, Siddharth Gupta and Veena Padmanabhan are Electrical Engineering majors at Rice University, Houston, TX. They are all in their third year of study and are all currently pursuing Signals and Systems as their specialization.

Acknowledgements

During our project, we were extremely fortunate to receive invaluable assistance from several people. We can not thank you enough:

Dr. Richard Baraniuk, our professor, for pointing us in the right direction time and again.

Ilan Goodman, for providing his motion detection code, numerous explanations, and his intuition analyzing our early results.

Marco Duarte, for his explanation of compressed sensing.

Mark Davenport, for his explanation of support vector machines.

Bruce Flinchbaugh, Texas Instruments, for sources on traditional intelligent motion detection.